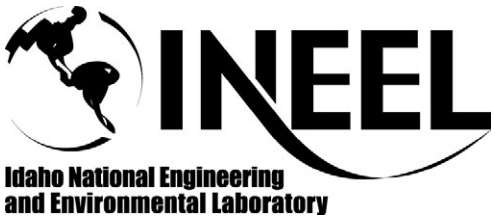

Idaho National Engineering
and Environmental Laboratory

INEEL/CON-04-01971
PREPRINT

Experiences in the Performance Analysis and Optimization of a Deterministic Radiation Transport Code on the Cray SV1

Peter Cebull

May 17 – 21, 2004

Cray User Group Meeting

Experiences in the Performance Analysis and Optimization of a Deterministic Radiation Transport Code on the Cray SV1

Peter Cebull, *Idaho National Engineering & Environmental Laboratory*

ABSTRACT: *The Attila radiation transport code, which solves the Boltzmann neutron transport equation on three-dimensional unstructured tetrahedral meshes, was ported to a Cray SV1. Cray's performance analysis tools pointed to two subroutines that together accounted for 80%-90% of the total CPU time. Source code modifications were performed to enable vectorization of the most significant loops, to correct unfavorable strides through memory, and to replace a conjugate gradient solver subroutine with a call to the Cray Scientific Library. These optimizations resulted in a speedup of 7.79 for the INEEL's largest ATR model. Parallel scalability of the OpenMP version of the code is also discussed, and timing results are given for other non-vector platforms.*

KEYWORDS: *Attila, Cray, SV1, radiation transport, vectorize, ATR*

1. Introduction

1.1 INEEL

The Idaho National Engineering and Environmental Laboratory (INEEL) is a science-based, applied engineering national laboratory dedicated to supporting the U.S. Department of Energy's (DOE's) missions in environment, energy, science and national defense. Established in 1949 as the National Reactor Testing Station and for many years the site of the largest concentration of nuclear reactors in the world, the laboratory's mission has changed and broadened into other areas over the years, such as biotechnology, energy and materials research, and conservation and renewable energy.

For many years the lead laboratory for the DOE Office of Environmental Management, in the summer of 2002 the INEEL was given new mission direction from the Secretary of Energy. Designated the lead laboratory for the Office of Nuclear Energy, Science, and Technology in partnership with Argonne National Laboratory, the new Idaho National Laboratory will return to its historical roots as the nation's leading center of nuclear energy research and development.

1.2 ATR

Of the 52 nuclear reactors built at the laboratory over the years, the Advanced Test Reactor (ATR) is one of three still in operation (Fig. 1). The world's premier test reactor, it is a unique facility that is used to create a wide range of reactor environments in which the effects of radiation on materials and fuels may be studied. Information that would normally require years of irradiation can be obtained after only weeks or months of exposure in the ATR's high flux environment.

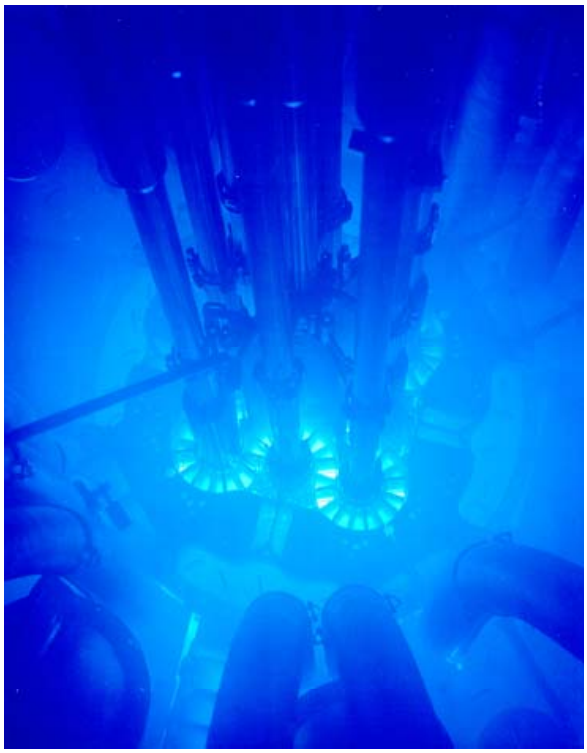

Figure 1: The Advanced Test Reactor in operation. The bright blue region that curves around the cylindrical test sections contains the fuel elements.

Although the primary user of the ATR is the Naval Nuclear Propulsion Program, there are other government, commercial, and foreign users. Its unique four-leaf-clover design provides nine main test spaces. Other smaller test spaces allow additional experiments to be conducted independently. These smaller spaces are routinely used for the production of medical and industrial isotopes.

1.3 Motivation

In early 2003, the INEEL obtained three Cray SV1s from the National Energy Research Scientific Computing Center (NERSC). One of the author's first tasks was to evaluate the applications being run at the laboratory and decide which ones would perform well on a vector architecture. Any codes that could be moved to the Crays would free up cycles on the other non-vector machines, which were becoming rather heavily loaded.

The laboratory's main compute servers consist of: 1) an SGI Origin 3800 with 64 MIPS R12k 400 MHz processors, 2) two Sun Fire 4800s, each with 12 UltraSPARC 750 MHz processors, and 3) three Linux clusters with AMD or Intel processors, containing up to 42 dual-processor nodes. These platforms are often largely consumed by MCNP processes. MCNP [1] is a general-purpose radiation transport code that uses Monte Carlo methods to solve the transport equation for neutrons, electrons, and photons. It is one of the workhorse applications used by nuclear engineers at the laboratory and probably consumes more CPU cycles at the INEEL than any other application. MCNP runs in parallel using either PVM or MPI, and it is not unusual for a run to consume 8 or 10 processors and run for several days.

MCNP is known to be a poor performer on vector architectures [1,2], so it is better suited to run on the clusters or the SGI Origin. However, Attila, another radiation transport code being evaluated for use at the laboratory, showed much more promise as a potential candidate for the SV1. The largest Attila ATR models developed at the INEEL contain 2.5 million computational elements and consume more than 7.5 Gbytes of memory. Prior to this porting effort, these large models were run on a single node of the AMD Opteron cluster, specially outfitted with enough extra RAM to hold them. This not only consumed resources that are better utilized by cluster-friendly codes like MCNP, it also limited the users to one run at a time. If Attila could be successfully ported to the high bandwidth, large memory (and more lightly utilized) Crays, the engineers would be able to make multiple runs simultaneously, hopefully with shorter run times, and free up computational resources for other applications.

One added benefit gained from this project was the experience gained in vector programming and performance tuning. The laboratory had a Cray Y-MP until the early 1990s, but it was removed and never replaced. The vector programming expertise gained up to that point has largely been forgotten or lost to attrition over the years. As the laboratory prepares itself to become the leading center for nuclear energy research and development, a renewed emphasis on high performance computing will likely require new expertise in the efficient utilization of vector platforms, among end users as well as support staff.

2. Attila

2.1 Description

Attila is a radiation transport code developed by the Transport Methods Group (CCS-4) at Los Alamos National Laboratory [3]. Originally intended for internal research purposes only, it was later spun off by Radion Technologies, a for-profit company that develops and markets the commercial version [4]. It consists of approximately 45,000 lines of Fortran 90, with some C preprocessor commands.

Attila is a deterministic radiation transport code, meaning it solves the Boltzmann transport equation analytically on a computational mesh. This is in contrast to statistical methods such as Monte Carlo, in which explicit transport equations are not solved, but rather large numbers of particles are tracked individually and their average behavior tallied. One benefit of using a deterministic code is that the solution is calculated everywhere on the computational mesh. Monte Carlo supplies information only for specific locations, which must be determined and specified prior to the calculation.

Attila is a three-dimensional discrete ordinate (S_N) code, which solves the first order form of the steady state transport equation on an unstructured, tetrahedral mesh. The S_N approach discretizes the transport equation's angular dependence into a set of solid angles. A multigroup energy discretization is used to represent energy dependence, and a linear discontinuous finite element method is used for spatial discretization. This results in four angular flux unknowns per computational element [5].

This system of equations is solved using a source iteration technique, meaning a series of source (outer) iterations are performed until the flux solution is converged. For optically thick problems dominated by scattering, source iterations can take a prohibitively long time to converge, thus some type of acceleration is required. Attila uses a modified diffusion synthetic acceleration (DSA) method to greatly reduce the number of source iterations required.

One final note should be mentioned. The optimizations described in this paper were studied over a period of several months and three versions of Attila. However, the discussion of code structure refers to Attila Version 3.12.0, and all timing runs were performed again as necessary using this version to maintain consistency.

2.2 Code Structure

The general structure of the Attila solver is outlined in Figures 2 and 3. Subroutine OUTER performs the outer source iteration loop, which continues until the specified convergence criterion has been met. Within this outer loop

is another loop over the number of energy groups being represented.

For each energy group, a call to subroutine INNER is made. This performs a within-group scattering iteration, which continues until the flux solution converges. Each iteration performs a call to SOLVE_FO and, if required, a call to DSA. Subroutine SOLVE_FO solves the transport equation for the new angular flux moments in each element. Subroutine DSA performs the Diffusion Synthetic Acceleration scheme mentioned earlier. DSA can be turned on or off through user input. Its use will not change the solution, only the time to converge.

```
SUBROUTINE OUTER

DO until converged
   DO 1, NGROUPS
         .
         .
      CALL INNER
         .
         .
   END DO
END DO
```

Figure 2: Subroutine OUTER controls the outer (fission source) iterations. NGROUPS is the number of energy groups being modelled.

```
SUBROUTINE INNER

DO until converged
     .
     .
   CALL SOLVE_FO
     .
     .
   CALL DSA
     .
     .
END DO
```

Figure 3: Subroutine INNER performs the inner (within group) scattering source iteration. SOLVE_FO solves the transport equation for new angular flux moments, and DSA executes the Diffusion Synthetic Acceleration scheme, which speeds up the iteration process.

3. Performance Analysis

3.1 Test Case Descriptions

The first step in optimizing Attila was the selection of some appropriate test cases to run Cray's performance analysis tools on. The Attila distribution came with example input for a nuclear reactor benchmark case. The Attila model, referred to here as "NEACRP," represents a $1/8^{th}$ reactor core. It uses two energy groups and 24 angles (discrete ordinates) in the transport calculation, and contains 3,932 elements in the computational mesh. This model is relatively small, but it is good for initial performance tuning due to its short run time.

The other major test case to be evaluated is the largest ATR model built to date. It contains 2,528,838 elements, four energy groups, and 24 angles. While too large to run in a reasonable amount of time using the profiling tools (at least prior to tuning), this case is most representative of the type and size that will be used most often to model the ATR.

Certain key characteristics of these models are summarized in Table 1. The variable *npoints* represents the number of cell vertices in the computational mesh (four vertices per tetrahedron), and *ncells* is the number of tetrahedral elements. For problems in which DSA has been selected, *ncoeffs* contains the number of non-zero coefficients in the upper triangular portion of the DSA operator. This number turns out to have a large impact on the overall run time and the amount of time spent in the DSA subroutine.

Two other smaller ATR models are also listed in Table 1. These models were run in order to check results after various optimizations were done, but their results are not discussed in this paper.

Table 1. Test problems and some of their key parameters

Model	npoints	ncells	ncoeffs
NEACRP	827	3,932	5,884
ATR_small	37,016	110,823	221,868
ATR_med	244,275	1,361,682	1,884,710
ATR_large	439,695	2,528,838	3,444,470

3.2 Analysis Results

The NEACRP example problem was used for the initial performance analysis since it could be run quickly. On the SGI Origin the NEACRP problem ran in 34.1 s. The original unoptimized Cray version of Attila took 99.7 s to run on the SV1. The group 0 hardware counters indicated that the overall code was achieving only 21.78 MFLOPS and 80.49 MIPS, for a MFLOPS/MIPS ratio of only 0.27. This indicated a very low degree of vectorization for the code as a whole.

Next, subroutine level timing was performed by recompiling Attila with the flowtrace feature enabled. This indicated that the subroutine that solves the transport

equation, SOLVE_FO, was responsible for 83.9% of the CPU time (Fig. 4). Clearly, this was the target for the optimization effort, as any improvement here would have a significant effect on the overall run time.

The next two most significant subroutines were CGD and DSA, which together accounted for only 6.6% of the CPU time. DSA, as described previously, performs the Diffusion Synthetic Acceleration algorithm. CGD is simply a preconditioned conjugate gradient solver based on the algorithm described in Ref. 6. It is called from within DSA to solve the DSA system of equations.

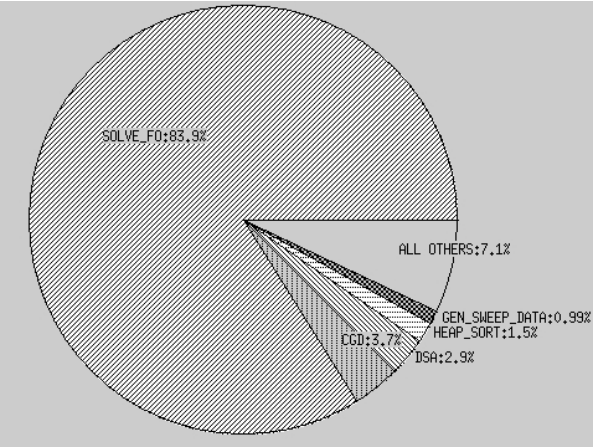

Figure 4: Flowview output for the unoptimized version of Attila. These results were obtained by running the NEACRP example problem.

A closer look at SOLVE_FO revealed the structure outlined in Fig. 5. An outer loop cycles over the number of angles (24 for these test cases). For each angle, a predetermined number of sweeps is performed. During each sweep the transport equations are solved for each of three element types: those with one side visible, two sides visible, or three sides visible.

The transport sweeps can be visualized by imagining the sweeps as wavefronts travelling across the entire mesh, roughly in the direction of the current angle [7]. This means that *ncells* 4 x 4 matrices are solved for each angle, or put another way, the transport equations for each element are solved 24 times, once for each direction.

This structure has implications from an optimization point of view. Rather than having a single, high trip count loop over the total number of mesh elements, the elements are instead broken up over several loops. Each sweep in the sweep loop only solves the transport equations for a subset of the total number of elements. Furthermore, this subset is further broken down into the three cell types across the three inner loops. This reduces the potential average vector length for the inner loops, but this effect should not be as significant for the mesh sizes typical of the ATR models described above.

4. Optimization

4.1 Vectorization of SOLVE_FO

Further profiling showed that the three inner loops in SOLVE_FO were responsible for most of the CPU time consumed in that subroutine, so they were obvious candidates for vectorization. Examination of the loopmark listing showed that none of them was being vectorized, although at least one reason was obvious, calls to subroutine LU4 in each loop.

```
SUBROUTINE SOLVE_FO

DO over angles

  DO over sweeps in angle

  ┌─DO 1-side visible cells
  │      .
  │    CALL LU4
  │      .
  └─END DO

  ┌─DO 2-sides visible cells
  │      .
  │    CALL LU4
  │      .
  └─END DO

  ┌─DO 3-sides visible cells
  │      .
  │    CALL LU4
  │      .
  └─END DO

  END DO

END DO
```

Figure 5: Basic structure of subroutine SOLVE_FO. The three interior loops, which solve the transport equations for each element, consumed the most CPU cycles and needed to be vectorized.

The 4 x 4 coefficient matrix for its associated element is constructed within each of the three inner loops. Subroutine LU4 is called to perform an LU decomposition of the coefficient matrix, and then the system of equations is solved by forward and back substitution to yield the new angular fluxes. LU4 uses Crout's algorithm with partial pivoting.

After inlining the calls to LU4, the compiler still would not vectorize the loops because of recurrences in three arrays: AMAT(*nvrtx,nvrtx*), IB(*nvrtx*), and PSI(*nvrtx*), where *nvrtx* is the number of vertices (4 for tetrahedra). AMAT is the coefficient matrix for the current element, IB is the output vector from LU4 which records the row permutations created by the partial pivoting, and PSI contains the new angular fluxes for each vertex.

One approach which seems to work is to promote these arrays by adding a new dimension, so that they are dimensioned as AMAT(*nvrtx,nvrtx,ncells*), IB(*nvrtx,ncells*), and PSI(*nvrtx,ncells*). A CONCURRENT directive was also required to convince the compiler to vectorize the loops, and a PREFERVECTOR directive was needed to prevent less significant inner loops from vectorizing instead. The modified arrays are now indexed by *i*, the loop index, and the appropriate array sections are passed during the call to LU4, instead if the original arrays:

```
call lu4(ier, AMAT(:,:,i), IB(:,i))
```

This strategy comes at the cost of a lot of extra memory used. For example, for the ATR_large test case with over 2.5 million elements, the extra dimension adds over 40 million words of required storage for these three arrays. Still, the performance payoff is large, even with the added storage.

A closer look at the loopmark listing revealed that, in a different section of SOLVE_FO, the wrong loop was vectorizing in a loop nest that accumulates angular flux moments. An inner loop with a trip count of four was vectorizing, so another PREFERVECTOR directive was used to get the outer loop to vectorize over the number of cells.

After both of these modifications were done, SOLVE_FO went from 22.24 MFLOPS to 55.01 MFLOPS, a 147% increase. The average vector length went from 9.32 to 21.82. This is a good improvement, although with the relatively small mesh size the vector performance still suffers. Overall, wall clock time decreased from 99.7 s to 63.5 s, a speedup of 1.57. Floating point performance for the code as a whole improved from 21.78 MFLOPS to 43.32 MFLOPS.

4.2 Memory Stride Issues

Although vectorization of the three loops in SOLVE_FO was a big first step, it still was not showing the kind of performance that one would expect from a well-vectorized subroutine. MFLOPS rates still seemed low, even considering the small problem size. The performance analysis tools gave indications of possible memory access problems. For instance, in runs of the NEACRP problem the group 0 counters showed the percent of clock periods (%CP) in SOLVE_FO holding issue to be 80.83%. Furthermore, group 1 counters showed the percent of all CPs waiting on memory ports in SOLVE_FO to be 66.81%. A partial run of the ATR_med test case showed 91.74% of CPs holding issue, and 83.11% of all CPs waiting on memory ports.

An examination of the array declarations in SOLVE_FO revealed that in several arrays (including AMAT and IB), either the first or first two leading dimensions were four, and loop iterations were being performed on the next dimension. This was resulting in strides of either four or sixteen, both powers of two (thereby hurting performance).

The initial reaction was to rearrange the array dimensions such that the loop indices were iterating on the leading dimension, giving a unit stride through memory. This approach was eventually abandoned, because the three loops mentioned in the last section quit vectorizing. This was apparently because a non-contiguous section of memory was being passed to LU4 [8] (e.g., `AMAT(:,:,i)` is a contiguous array section, whereas `AMAT(i,:,:)` is not). The extra processes involved in passing the non-contiguous sections to LU4 were enough to prevent the current inliner and optimizer from vectorizing the loops.

Another, simpler technique was used instead. The strides were made odd by adding one to the leading dimensions. This resulted in a net improvement, although again at the cost of more memory being consumed. The actual number of CPs being spent waiting on memory ports dropped 35% (for the NEACRP case). Floating point performance in SOLVE_FO improved from 55.01 MFLOPS to 75.10 MFLOPS, and the overall wall clock time dropped another 10% from 63.5 s to 56.9 s. A final look at subroutine timings showed that SOLVE_FO now accounted for 61.2% of total CPU time, CGD 8.6%, and DSA 7.1%.

4.3 Vectorization of DSA

Once subroutine SOLVE_FO seemed to be tuned pretty well, it became time to try the performance tools on the largest test case, ATR_large. Running the optimized version of Attila containing the SOLVE_FO vectorization and memory stride changes produced the profile shown in Fig. 6. On a mesh of this size, the vast majority of the computational effort has been shifted to subroutine CGD, which solves the DSA system of equations. SOLVE_FO now used only 9.6% of the total CPU time.

Subroutine DSA only accounted for 1.8% of the CPU time, but a quick look at the loopmark listing revealed two loops that appeared to be vectorizable, so this routine was tackled prior to attempting CGD. The structure of DSA is relatively simple. There is an initial loop over *ncells* that creates the DSA source vector, followed by a call to CGD (which solves the DSA system), followed by a final loop over *ncells* that solves for the DSA scalar flux correction.

The only thing preventing vectorization of the initial loop was the accumulation of an array at the bottom in which indirect addressing was used. The loop would vectorize with a CONCURRENT directive, but subscript collisions within the vector length resulted in code failure. This problem was solved by breaking the accumulation

portion into a separate loop, allowing the remainder of the loop to vectorize.

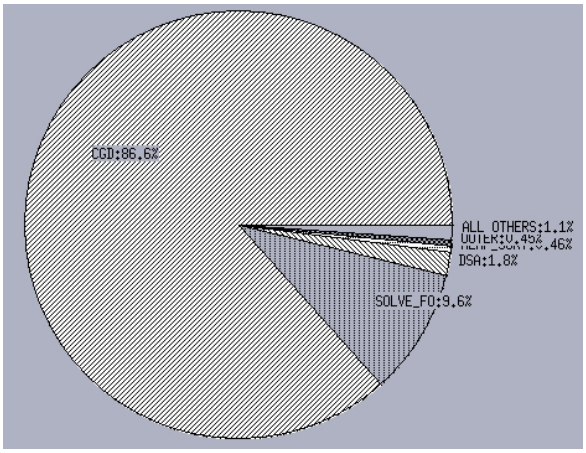

Figure 6: Perfview output after running the large ATR model with the optimized version of SOLVE_FO. The much larger model spends most of its effort in the conjugate gradient solver, CGD.

The final loop in DSA is similar to the inner loops in SOLVE_FO. A 4 x 4 system of equations is set up and solved for each cell, except here the loop is over *ncells* and is not split up into separate loops. An additional dimension had to be added to the coefficient matrix and solution vector to eliminate recurrences, as was done in SOLVE_FO. This time, a PREFERVECTOR directive was sufficient to get the loop to vectorize, so no CONCURRENT directive was required.

Vectorizing the two main loops in DSA increased performance from 22.4 MFLOPS to 71.0 MFLOPS and increased the average vector length from 59.2 to 64.

4.4 Conjugate Gradient Solver

Subroutine CGD obviously needed to be improved in order for Attila to perform well enough to be run on the SV1. It was achieving only 21.8 MFLOPS and consuming 87.6% of the total CPU time. Unfortunately, there was no way to get it to vectorize as it was written. A complete replacement would be necessary in order to get acceptable performance.

As luck would have it, the entire CGD subroutine could be replaced by a call to SITRSOL in the Cray Scientific Library. Subroutine CGD is simply a preconditioned conjugate gradient solver, and the same functionality is supplied in SITRSOL [9,10]. The call to CGD was commented out and replaced by calls to DFAULTS (to assign default values to the parameter arguments for SITRSOL) and SITRSOL. Diagonal (Jacobi) preconditioning was selected to conform to the preconditioning option available in Attila.

The magnitude of the speedup achieved with this modification can best be seen by looking at the average time per call to subroutine DSA, including the time spent in the conjugate gradient solver. Using the CGD solver, the average time per call for DSA was 7.06 s, and the time per call to CGD was 1100 s, for a total of 1,107 s per call. After replacing CGD with SITRSOL, the time per call to DSA was 102 s (including time spent in SITRSOL). This represents an order of magnitude improvement in the time spent solving the DSA equations. Wall clock time for the entire run went from 99.0 hours to 19.8 hours for a speedup of five.

5. Results

5.1 NEACRP

The overall speedup results for this model were not particularly impressive, although they are a definite improvement. The floating point performance for the entire code increased from 21.8 MFLOPS to 59.5 MFLOPS. The wall clock time decreased from 99.3 s to 52.1 s, for a speedup of 1.91. The ratio of MFLOPS to MIPS increased from 0.27 to 1.09, showing a modest improvement in vector performance.

As mentioned previously, this model is really too small to show good performance in Attila. A relatively large amount of time is spent performing I/O while input files are being read and output files are being created, compared to the amount of time actually spent performing the transport calculations.

Fig. 7 shows the speedup achieved with each of the optimizations described in this paper. Most of the improvement for a model of this size is seen by vectoring SOLVE_FO. Not enough time is spent in the DSA solving routines for the latter optimizations to have as significant an effect.

Fig. 8 shows the wall clock run times for three platforms: the 300 MHz Cray SV1, a 400 MHz SGI Origin 3800, and a 1.8 GHz Opteron-based PC. Although much improved, the SV1 run time for a model this small is still not comparable to that achieved with a modern desktop PC.

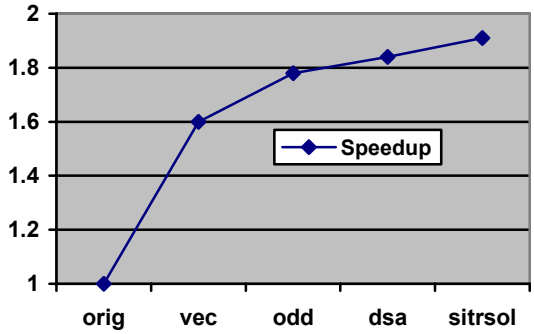

Figure 7: Speedup achieved during each stage of the optimization process running the NEACRP model. The labels on the x-axis indicate the original, vectorized SOLVE_FO, odd memory stride, vectorized DSA, and SITRSOL versions.

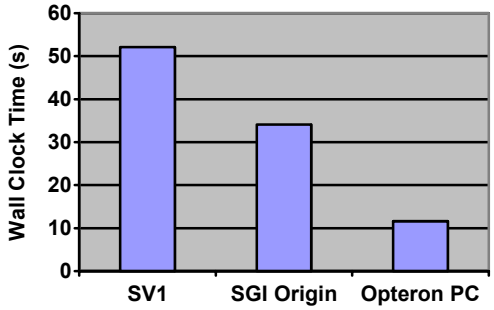

Figure 8: Wall clock run times for the NEACRP model. The SV1 results are with the final optimized version. SGI and PC results are for the original unmodified source.

5.2 Large ATR Model

This model, which is typical of the kind that will be used at the INEEL, showed a much more dramatic increase in performance. The code improved from 22.3 MFLOPS for the unaltered version to 136.6 MFLOPS with the final, optimized version. The final MFLOPS/MIPS ratio was 4.34. The most dramatic improvement was in wall clock time. It decreased from 154.5 hrs (6.4 days) to 19.8 hrs, a speedup of 7.79.

The speedup graph in Fig. 9 shows that the largest single improvement was obtained by replacing the generic conjugate gradient solver CGD with the optimized version found in the Cray Scientific Library. With so much time being spent in the DSA equation solver, it is imperative that this portion of the code be well vectorized. This is a clear demonstration of the utility of vendor-optimized library routines as compared to hand-coded versions, especially for a vector platform.

Wall clock times for the Cray and Opteron PC are shown in Fig. 10. The SV1 is slightly faster when run on a single Single Streaming Processor (SSP).

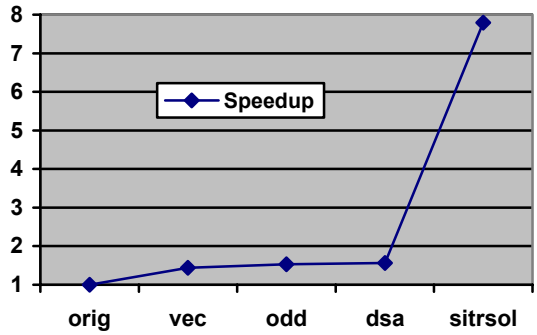

Figure 9: Speedup achieved during each stage of the optimization process running the large ATR model. For a model of this size, almost all of the speedup was obtained by using the optimized solver in the Cray Scientific Library.

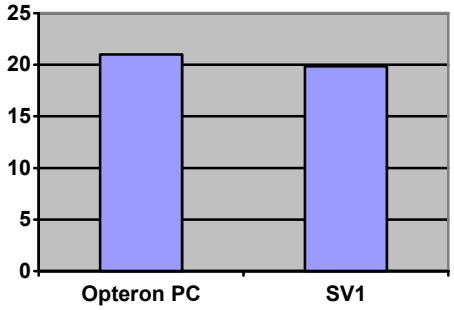

Figure 10: Wall clock run times for the large ATR model. SV1 results are with the final optimized version. The SV1 is slightly faster for a single-processor run, approximately 20 hrs vs. 21 hrs.

5.3 Parallel Performance

Attila was originally intended to be run on single-processor desktop PCs or workstations [3], and this mode of operation is perfectly adequate for most "typical" models, such as the NEACRP test case. However, full-core models of reactors such as the ATR can quickly overwhelm the capabilities of these machines. In Version 3.12.0 of Attila, the developers began looking at ways to run the code in parallel.

They began in an incremental fashion, using OpenMP directives to parallelize two loops. One loop is in subroutine OUTER, but it is in a portion of code that is not executed an appreciable amount of time, at least for the ATR models. The one major loop that was parallelized by the developers is the outer loop in SOLVE_FO over the number of angles (see Fig. 5). An OpenMP PARALLEL DO directive was used with a static schedule to split the angles over the number of threads.

Parallelizing this portion of code will help increase code speed most for those problems that spend a lot of time in SOLVE_FO, like the NEACRP test case. Unfortunately, ATR models of the size being run at the INEEL will not show appreciable speedup, since most of the time is spent in DSA.

It is for this reason that the use of the SITRSOL solver provides another benefit beyond the single-process speedup already observed. Since SITRSOL is multitasked as well as vectorized [9], a greater parallel scalability is seen by running the Cray version of Attila, especially for large problems. Fig. 11 shows the scalability of the Cray and Opteron-based PC versions when running the ATR_med model.

Not only is the Opteron PC limited by low parallel scalability, the user is also limited to two processors, since it is a dual-processor node. The INEEL SV1s are only limited by the current license to four processors. Fig. 12 shows the wall clock times for both machines, for single- and multiple-processor runs of the ATR_large model.

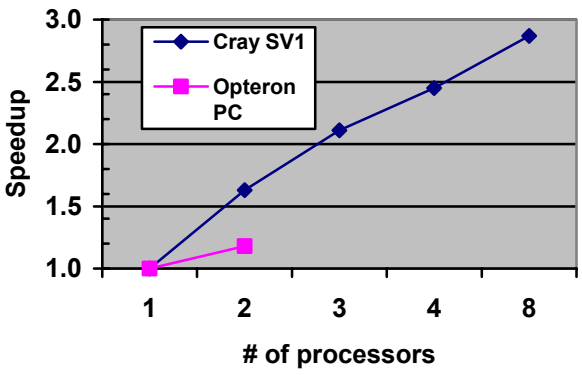

Figure 11: Parallel scalability of the medium ATR model. The Cray version scales better, mostly due to the parallelism built into the SITRSOL solver.

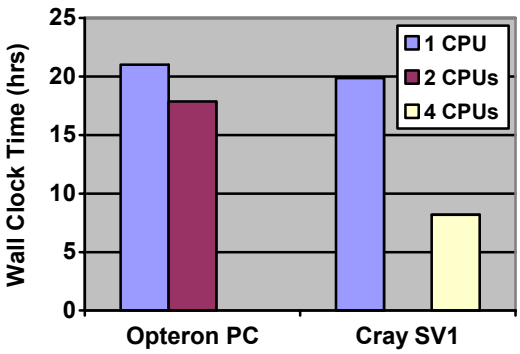

Figure 12: The large ATR model will complete 2.2 times faster on the Cray's four processors than the Opteron's two.

6. Conclusions

The results of this optimization effort show that Attila can be modified to produce an acceptable level of performance on the Cray SV1. The SITRSOL sparse matrix solver in the Cray Scientific Library produced the greatest speedup for the large ATR model, due to the extremely large DSA system of equations that had to be solved. In addition, the ability of SITRSOL to run in parallel resulted in greater scalability in the Cray version of Attila.

The bottom line is that the nuclear engineers at the INEEL are now able to run their largest models in less than half the time than they previously could. They also have the capability to run multiple jobs simultaneously.

There is probably more that can be done to optimize Attila, and there very well may be better ways of vectorizing the subroutines than were used here. Now that Attila has been demonstrated to run well on the Crays, a larger effort to optimize the code is likely to occur.

Acknowledgments

The author would like to thank Bill Long and Jim Maltby of Cray Inc. for their assistance. Bill provided valuable suggestions for speeding up and vectorizing the calls to the LU factorization routine, and Jim spent several hours going over the Attila source during a site visit to the INEEL.

Work supported by the U.S. Department of Energy under DOE Idaho Operations Office Contract DE-AC07-99ID13727.

References

1. Judith F. Briesmeister, ed., "MCNP™ – A General Monte Carlo N-Particle Transport Code," LA-13709-M, March 2000.

2. Jim Maltby, "Cray SV1 Performance Tutorial," presented at the INEEL, June 10, 2003.

3. Attila Radiation Transport Code Home Page, http://www.lanl.gov/attila/index.html.

4. Radion Technologies Home Page, http://www.radiative.com.

5. J. M. McGhee and T. A. Wareing, "Attila Version 3: User's Manual," 17 December 2003.

6. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd edition, Johns Hopkins University Press, Baltimore, MD, 1989.

7. Todd A. Wareing, et al., "Discontinuous Finite Element S_N Methods on Three-Dimensional Unstructured Grids," *Nuclear Science and Engineering*, **138**, pp. 256-268 (2001).

8. E-mail correspondence from Bill Long of Cray Inc., in response to Request for Technical Assistance #3001, 15 Aug 2003.

9. "Scientific Libraries User's Guide," S-2151-36, Cray Inc., 2002.

10. M. A. Heroux et al., "A Parallel Preconditioned Conjugate Gradient Package for Solving Sparse Linear Systems on a Cray Y-MP," *Applied Numerical Mathematics*, **8**, pp. 93-115 (1991).

About the Author

Peter Cebull is an Advisory Engineer on the HPC/visualization team at the INEEL. He has a background in nuclear engineering code maintenance and development and now serves as a Cray subject matter expert in support of INEEL staff. He can be reached at INEEL, P.O. Box 1625, MS 3605, Idaho Falls, ID 83415-3605 USA, Phone: 208-526-1909, E-Mail: cebupp@inel.gov.